# Stable Grounded Inference in Flexible Resource Scheduling

Paul Morris and John Bresina

NASA Ames Research Center Moffett Field, California, U.S.A.

#### Abstract

Systematic temporally flexible solving has proved useful in scheduling with temporal and resource constraints. However, determining resource flaws in a flexible framework is a complex task even for activities with instantaneous resource impacts, and becomes more daunting for linear or more complex impacts. Plan stability is also important for many applications; as new constraints arise, the changes to an existing plan to accommodate them should be minimized to a degree consistent with rapid solving. In this paper we present a method that uses an evolving grounded solution to guide a flexible resource solving process. To promote stability, the successive grounded solutions satisfy a minimal perturbation criterion. We also discuss an encoding of state constraints as numeric resource transactions and some theoretical implications for state reasoning.

#### Introduction

Combining inference and search in the most effective way is a critical aspect of solving combinatorial problems. In many cases there is a trade-off regarding how much emphasis to put on inference versus search when solving a problem. Inference can support a more intelligent search, but it comes with an overhead cost. Thus, the effectiveness of searching a smaller space more slowly must be weighed against that of searching a larger space more quickly. As a consequence, effective inference mechanisms must be fast in relation to the complexity of the search being conducted.

Performance is not the only consideration when designing algorithms for diverse applications. Extensibility is also important. Thus, we would like inference mechanisms to be straightforward so that they can be easily extended to handle more complex problem environments. Robustness is another consideration. We would like solutions to have built-in flexibility to cope with execution uncertainty. There is another potential tradeoff here because obtaining flexible solutions may require more elaborate inference mechanisms. Fortunately, these conflicts are not inevitable; sometimes it is possible to find approaches that simultaneously achieve several of these objectives.

These considerations apply to resource scheduling problems with *flexible* temporal constraints. For these problems, activity start times, end times, and durations are represented Javier Barreiro SGT Inc. NASA Ames Research Center

Moffett Field, California, U.S.A.

as numerical intervals instead of definite (also described as "grounded") numeric values. Temporal flexibility is useful for dealing with execution uncertainty (Muscettola 2004); therefore, for many problems a flexible solution is preferred over a grounded one. Flexible solutions can be generated as follows: the temporal variables and constraints are represented as a Simple Temporal Network (STN) (Dechter, Meiri, and Pearl 1991); temporally flexible resource levels are computed using the resource envelope algorithm developed in (Muscettola 2004); and a search algorithm, using the envelopes to check feasibility, imposes precedence constraints to generate flexible solutions.

However, operating completely in flexible space using these mechanisms is costly and complicated. In particular, computing the resource envelopes requires maximum flow calculations. The complexity is magnified if the resource model is extended to activities that have linear resource impact (Frank and Morris 2007), where only a pseudo-polynomial algorithm is known for computing the envelopes, and it is unclear how the envelope method could be extended beyond that to complex nonlinear resources, such as the power from solar panels on a Mars rover, which follows a sine curve with respect to time of day. As a consequence, when confronted with these problems, practitioners often resort to reformulating the problem as a grounded one, without temporal flexibility.

In this paper, we explore an approach that generates flexible schedules within a framework that uses simple resource calculations. We do this by performing grounded inference on resources in the context of flexible temporal reasoning. Thus, the resource calculations are simple, but we still end up with a flexible solution.

One way of approaching this is demonstrated by a previous paper (Cesta, Oddi, and Smith 1998). It is known that an assignment of lower-bound values to each of the variables in a Simple Temporal Network (STN) constitutes a grounded schedule that satisfies the temporal constraints. In the Cesta et al. paper, this solution is used to compute a resource profile, and the flexible solving is restricted to adding STN constraints to remove flaws observed within the profile. The outcome is an augmented STN. However, only the lower-bound schedule is guaranteed to be free of resource flaws. By contrast, using the envelope approach (Muscettola 2004) to detect flaws results in an STN where every schedule is free of resource flaws. The (Cesta, Oddi, and Smith 1998) paper compensates for this by a post-processing step: it chains the activities that use a resource according to their order in the lower-bound schedule. The result is a flexible solution (Cesta, Oddi, and Smith 1998).

However, there is an issue with respect to using lowerbounds for grounded inference. In many applications, especially in a mixed-initiative setting (Bresina and Morris 2007) where plans are evolved, and in plan repair (Yoon, Fern, and Givan 2007), plan stability is an important consideration. That is, it is desirable that a plan change as little as practicable when accommodating modifications and new constraints. In this framework, the mixed-initiative operator sees only a grounded schedule, called a reference sched*ule*. Initially, this is provided by the user and reflects general preferences. As planning proceeds, the user may make additional changes. The schedule may also need modifications to satisfy constraints, but these should try to minimize disruptions to the previous settings. The lower-bound schedule has no memory of previous schedules, so it does not meet this requirement. In particular, it does not reflect direct changes to the schedule made by the operator.

In previous work (Bresina and Morris 2007), a solution grounding algorithm was used to provide the evolving reference schedule according to a minimum perturbation criterion. However, this was applied after full runs of a flexible solver doing mutual exclusion reasoning; it was not used for flaw detection. In this paper we report on a new approach that solves resource problems using the reference schedule for detecting resource flaws. This requires that the reference schedule be recomputed after each flaw resolution step. However, the solution grounding algorithm used in (Bresina and Morris 2007) involved O(N) calls to Dijkstra's Algorithm (Cormen, Leiserson, and Rivest 1990), which is too costly for our present purpose. In this paper we present a new solution grounding algorithm with minimum perturbation properties that has aspects in common with upper and lower bounds, and needs only one Dijkstra call. Another difference is that the (Bresina and Morris 2007) algorithm was a greedy one whose results were dependent on the order in which timepoints were considered. The new algorithm is order-independent and has a global minimality property.

This work uses the more general producer/consumer resource model of (Muscettola 2004) rather than the reusable resource model of (Cesta, Oddi, and Smith 1998). This permits the grounded solution to be extended to a flexible solution in a way that retains less rigid constraints than the method of extracting chains. It also supports an encoding of state requirements and effects as resource transactions.

Local search algorithms also typically search in a grounded framework, but they are unable to show a problem is unsolvable. The approach here combines a complete systematic search with the simplicity and extendability of grounded reasoning.

In summary, we present an approach that

• Generates flexible solutions more efficiently by using lower complexity grounded inference on a reference schedule to detect resource flaws within a flexible temporal solving process.

- Addresses plan stability by using a novel grounding algorithm to derive the reference schedules while meeting minimum perturbation criteria.
- Can be extended to to other kinds of constraints; we will use state constraints as an example.
- Combines some of the benefits of local search with those of systematic search.

#### **Background and Solver**

Laborie (Laborie 2003) describes a simple but expressive formalism for scheduling problems called Resource Temporal Networks (RTNs). In brief, RTNs consist of a Simple Temporal Network (STN) as described in (Dechter, Meiri, and Pearl 1991), fixed instantaneous resource impacts (either production or consumption) for each timepoint, and constant upper and lower resource limits. Timepoints represent the start and end of activities. Instantaneous resource impacts are useful for modelling reusable resources that are allocated at the beginning of an activity and released at the end, such as power load on a planetary rover, or open-ended resource production and consumption by separate activities (e.g., switch-on, switch-off). Of particular interest, they can also be used to model state requirements and effects using the resource encoding described in (Morris and Bresina 2008).

More formally, an RTN defines a *problem* for a specific resource as a tuple  $\langle T, Q, l, u \rangle$ , where T is an STN, Q is a mapping from the timepoints of T to real numbers that specifies the incremental resource impact (positive for a production and negative for a consumption), and l and u are extended real numbers representing the lower and upper limits. Multiple resources can be modelled by separate RTNs.

Given an RTN, a *schedule* is a mapping S from the timepoints to real numbers that specifies the time S(x) at which each timepoint x occurs. A schedule has a *flaw* if the cumulative resource level at some time is outside the limits. The solver seeks to find a schedule that satisfies the temporal constraints and is free of flaws. Our solver has a basic cycle that (1) detects flaws in the current flexible plan, (2) selects a flaw to fix, (3) chooses a way of resolving the flaw, and (4) backtracks if the flaw cannot be fixed. Multiple RTNs can be solved together by interleaving the flaw resolution steps.

In our grounded inference approach, a reference schedule that satisfies the current temporal constraints is used to detect flaws in the flexible plan in step (1). These flaws constitute only a subset of all the flexible flaws, but are easy to calculate. The reference schedule is used in a forward simulation to construct a profile of the resource levels over time and collect the flaws. The selection in step (2) is based on heuristics. Step (3) adds a new constraint to the STN that may resolve the flaw. We describe this in more detail in the remainder of this section. Step (4) uses chronological backtracking to revise a previous choice from (3) if possible.

To facilitate the discussion, we assume that all flaws are lower-level flaws; upper level flaws can be handled symmetrically. We will refer to a timepoint whose resource impact is a consumption as a *consumer*, and one whose impact is a production as a *producer*. We will sometimes use time(x) in place of S(x) when a particular schedule S is understood.

Given an RTN and a reference schedule S, a flaw occurs at a time t if the cumulative resource level dips below the lower limit at that time. A consumer c is said to be a *culprit* with respect to the flaw if  $S(c) \leq t$ . The culprits are the timepoints that contribute to the resource shortfall at the flaw. A producer p is said to be a *savior* with respect to the flaw if S(p) > t. A producer that is not a savior is called a *helper*. Helpers mitigate the shortfall by positively contributing to the resource level at the flaw.

The *basin* of a flaw is the interval between the latest culprit and the earliest savior. Intuitively, to try to remove a flaw, we should move some culprit later and/or some savior earlier. To fully address the flaw, these movements need to cover the basin. We make that precise in the following fundamental lemma.

**Lemma 1** Suppose  $S_1$  and  $S_2$  are two schedules where  $S_1$  has a flaw but  $S_2$  has no flaws. Then there is a culprit timepoint c and a savior timepoint s for the flaw in  $S_1$  such that  $S_2(c) \ge S_2(s)$ .

# **Proof:**

Consider some flaw in  $S_1$  at time  $t_1$ . Let C and S be the respective culprit and savior sets. Let  $t_c = \max\{S_2(c) : c \in C\}$  and  $t_s = \min\{S_2(s) : s \in S\}$ . Suppose  $t_c < t_s$ . Then  $S_2(c) \leq t_c$  for every  $c \in C$  and  $S_2(s) \geq t_s > t_c$  for every  $s \in S$ . It follows that the resource value at  $t_c$  in  $S_2$  is no greater than the flawed value at  $t_1$  in  $S_1$ , which implies a flaw in  $S_2$  contrary to the statement of the theorem. Thus,  $t_c \geq t_s$  and so  $S_2(c) \geq S_2(s)$  for some culprit  $c \in C$  and savior  $s \in S$ .

From Lemma 1, we see that flaws can only be eliminated by schedule changes that result in time $(c) \ge time(s)$  where c is some culprit and s is some savior. Thus, some  $c \ge s$  constraint needs to be added in step (3), resulting in a new reference schedule where the culprit occurs at or after the savior. Other timepoints may also move because of constraints. The updating of the reference schedule will be discussed in more detail in the section on solution grounding.

A particular  $c \ge s$  constraint addition is not allowed if it is inconsistent with the existing STN constraints. If there are no allowed culprit/savior constraints, then the flaw is unfixable, and backtracking occurs in step (4).

Even if a culprit/savior constraint is allowed, there is no guarantee that the flaw shortfall is reduced in the new reference schedule; it may even be increased! The reason is: when a culprit is moved, a helper may be forced to move also because of constraints, which can negate the benefit of moving the culprit. It is necessary in general to allow such seemingly counterproductive moves for the sake of completeness of this simple solver strategy. Consider, for example, a case where two culprits  $c_1$  and  $c_2$ , each of resource value 3, are constrained to strictly precede a helper h with resource value 4. Since only one of these culprits can be moved in a single solver step, the first move will also move the helper and will increase the flaw shortfall by 1, but the second move will then decrease it by 3. A maximum flow calculation (Muscettola 2004) could be used to formulate

Boolean procedure solveFlaws(stn) Compute flaws in reference schedule; if no flaws then return true else choose a flaw; for each culprit/savior pair <c,s> if c >= s is consistent then Add constraint c >= s to stn; Update reference schedule; if solveFlaws(stn) then return true; Remove c >= s and add c < s to stn; Remove above added c < s constraints; return false;



smarter compound moves, but we are trying to avoid that with the grounded approach.

To assure non-redundancy in the search, we need the constraint choices in step (3) to be mutually exclusive; otherwise, multiple paths in the backtracking search could lead to the same search state. The  $c \ge s$  constraints are not *a priori* exclusive, but we can make them so by the simple expedient of adding the complementary c < s constraints as an alternative on backtracking.

For the step (3) choice, we order the culprit/savior pairs lexicographically with respect to closeness to the flaw, i.e.,  $\langle c_1, s_1 \rangle$  precedes  $\langle c_2, s_2 \rangle$  if time $(c_1) \rangle$  time $(c_2)$ , or time $(c_1) = \text{time}(c_2)$  and time $(s_1) \langle \text{time}(s_2)$ . This is in accord with the plan stability criterion.

The flaw solving algorithm described above is summarized in Figure 1, where stn is the temporal network including all added constraints. In the algorithm, the update of the reference schedule takes as input the *original* reference schedule (before the toplevel call to solveFlaws()), since this is what we wish to minimally perturb. The backtracking search is expressed as a combination of a for loop and a recursive call to solveFlaws.

If the algorithm terminates with success (true), the final reference schedule is output as the grounded solution. A failure termination (false) indicates there is no solution. Theorem 1 shows completeness and correctness. We note that completeness is not obvious because the algorithm only searches through producer/consumer orderings that arise from flaws in the successive reference schedules, not through all orderings consistent with the flexible plan. First we prove a lemma that is essentially a useful reformulation of lemma 1.

**Lemma 2** Consider two schedules  $S_1$  and  $S_2$ . Suppose for each consumer c, the set of producer predecessors of c in  $S_2$ is a superset of those in  $S_1$ . Then if  $S_1$  is without flaws, so is  $S_2$ .

#### **Proof:**

Suppose  $S_2$  has a flaw. By Lemma 1, there is a culprit c and a savior s with respect to this flaw such that  $S_1(c) \ge S_1(s)$ . But then c has an extra producer predecessor s in  $S_1$  that it does not have in  $S_2$ , contrary to the condition of this lemma.

**Theorem 1** The solveFlaws procedure terminates with a correct solution if one exists, otherwise with failure.

#### **Proof:**

Termination follows since: each step adds an ordering constraint; a particular ordering constraint cannot be added more than once; and the number of potential ordering constraints is bounded. If it terminates with success, then the reference schedule satisfies the temporal constraints and has no resource flaws; thus it is a correct solution.

To show completeness, assume the initial schedule I is flawed and suppose there is a solution schedule S. We need to show that the algorithm will then find some solution. Consider the set  $\mathcal{P}$  of consumer/producer pairs  $\langle c, p \rangle$  such that  $S(c) \geq S(p)$  but I(c) < I(p).

For the initial flaw selected by the algorithm, by Lemma 1, there must be at least one  $\langle c, p \rangle$  pair in  $\mathcal{P}$  such that c is a culprit and p is a savior. Assuming the algorithm does not otherwise find a solution, let  $\langle c_1, p_1 \rangle$  be the first such pair reached in the for loop. Since S is a solution, and none of the earlier rejected pairs in the for loop are in  $\mathcal{P}$ , the  $c_1 \geq p_1$  constraint is consistent with the previously added c < p backtrack constraints, and with the original constraints, so it will be added.

If the reference schedule I' after adding the constraint is still flawed, the same argument applies, and the algorithm will add another  $c \ge p$  constraint from the pairs in  $\mathcal{P}$ . If no other solution is found, the algorithm will eventually reach some reference schedule S' that respects all the precedences for  $\langle c, p \rangle$  pairs in  $\mathcal{P}$ . By Lemma 2, S' is a solution.

As discussed in the introduction, the reference schedule is a grounded solution, not a flexible one. For the producer/consumer resource model, extension of the grounded schedule to a flexible solution involves the following steps.

- 1. Discard the added solver constraints (but keep the original STN constraints).
- For each time(p) ≤ time(c) precedence in the grounded solution, where p is a producer and c a consumer, add a p ≤ c constraint to the STN.

Thus, we extract the producer/consumer precedences and add them to the STN constraints. The resulting STN is the flexible extension. Since the grounded solution has no flaws, it follows from lemma 2 that all the schedules for the flexible extension are flawless. This method of extension extracts weaker constraints from the grounded solution than the chain method of (Cesta, Oddi, and Smith 1998). For example, consider a grounded schedule where producers  $p_1, p_2$  and consumers  $c_1, c_2$  occur in the order time $(p_1) \leq \text{time}(c_1) \leq \text{time}(p_2) \leq \text{time}(c_2)$ . The chain method would require that every schedule obey that ordering, whereas the flexible extension here would permit additional orderings where both producers precede both consumers.

### **Solution Grounding**

We turn now to the *solution grounding* algorithm that, given a previous reference schedule and a set of temporal constraints, produces a new reference schedule that satisfies those constraints. The idea is that as constraints are added, the schedule is progressively modified to accommodate the new constraints.

In (Bresina and Morris 2007), a solution grounding algorithm is presented that requires O(N) propagations, where N is the number of timepoints. Each propagation is carried out using Dijkstra's algorithm (Cormen, Leiserson, and Rivest 1990), which has  $O(E+N \log N)$  complexity, where E is the number of edges. This is too costly for our current purposes since the reference schedule needs to be recomputed after each solver step. In this section, we introduce a new algorithm that needs only a single invocation of Dijkstra's algorithm.

To facilitate the discussion, we will refer to the input reference schedule as the *preferred time* schedule, and the output as simply the reference schedule. The objective is to have the reference times be close to the preferred times within the context of an efficient update algorithm. The preference functions are convex, so one might consider using the approach of (Morris et al. 2004). However, that requires solving a linear programming problem, which is not as efficient as the method presented here.

The new algorithm is inspired by an analogy with the computation of upper/lower time bounds for each timepoint in an STN (Dechter, Meiri, and Pearl 1991). There, to compute upper/lower bounds, the STN is augmented with a new virtual timepoint called the *origin*. Then the upper bounds are calculated as distances from the origin, and the lower bounds as negated distances to the origin. Both distance calculations use Dijkstra's algorithm. (See the discussion on Johnson's Algorithm in (Cormen, Leiserson, and Rivest 1990) for how Dijkstra's algorithm can be used to calculate distances in an STN where consistency has been verified.)

It should be noted that the reference schedule behaves a little differently from upper or lower bounds. As constraints are added, lower bounds can only increase while upper bounds can only decrease. However reference times may need to move in both directions. For example, suppose a timepoint x can range from 0 to 20, but has a preferred time of 10. If we add a constraint  $x \ge 15$ , then the new reference time should be 15, whereas if the added constraint is  $x \le 5$ , the new reference time should be 5.

Nevertheless, we proceed by adding a new virtual timepoint to the STN called the *refpoint*. The refpoint is distinct from the origin timepoint. Each preferred time value for a timepoint becomes an upper bound constraint from the refpoint to the timepoint (Figure 2). For example, suppose a timepoint x has a preferred time of 10. We add a constraint of the form x-refpoint  $\leq 10$ . The preferred time for the origin is 0 by convention, so a constraint (origin–refpoint)  $\leq 0$ is always added. The STN with these added constraints is the augmented STN. Note that in the distance graph of the augmented STN the refpoint has edges leaving it, but none entering it. It follows that the augmented STN is consistent if the original STN is, since any new negative cycle would need to pass through the refpoint. Similarly, the lower and upper bounds, defined in terms of paths to and from the origin, are unaltered in the augmented STN.

We can then calculate shortest path distances from the



Figure 2: Distance Graph with Refpoint



Figure 3: Corrected Preferred Times

refpoint to each timepoint, including the origin, using Dijkstra's algorithm in a similar way as for upper-bound calculations. For example, suppose x has a preferred time of 10, while y has a preferred time of 20 and a lower bound of 18.. If  $y - x \leq 5$  is added as a new constraint, then the shortest distance from the refpoint to x is 10, while the shortest distance to y is 15, as seen in Figure 2, where  $\mathcal{O}$  is the origin timepoint and  $\mathcal{R}$  is the refpoint.

We would like to define the reference time (reftime) of a timepoint x to be the shortest path distance from the refpoint to x. This distance can be no greater than the distance from the origin to x, i.e., the upper bound of x, because of the 0 edge from the refpoint to the origin. However, there is a problem: this distance may lie below the lower bound of the timepoint. In the example, the distance from  $\mathcal{R}$  to y is 15 while its lower bound is 18. Note the distance from  $\mathcal{R}$  to  $\mathcal{O}$  is -3, which is negative.

One might consider defining the reftime as distance (refpoint, x) – distance (refpoint, origin). However, this would mean the reftime of x could be affected by an offset determined by constraints that are independent of x, violating the stability criterion.

Note that any preferred time that is below the lower bound of its timepoint has to be increased at least to the lower bound in order to satisfy the constraints. This suggests correcting the preferred times before calculating the (shortest path) distances.

**Definition 1** The corrected preferred time of x (called the **preftime** for brevity) is defined by  $preftime(x) = \max\{lower-bound(x), preferred-time(x)\}.$ 

**Definition 2** *The* reference time of x (called the **reftime** for brevity) is reftime(x) = distance(refpoint, x) where the augmented STN has a constraint  $(y - refpoint) \leq preftime(y)$  for each timepoint y.

In the example, from  $y-x \le 5$  and  $y \ge 18$ , we can infer a lower bound of 13 for x. Thus, preftime(x) = 13, giving the corrected distance graph shown in Figure 3, which results in reftime(x) = 13 and reftime(y) = 18.

The following results show some nice properties of these definitions.

**Theorem 2** The reference schedule satisfies the temporal constraints.

# **Proof:**

It is easy to show that the distance values from the refpoint constitute a solution to the binary constraints in an STN. (A similar result is shown for the origin in (Dechter, Meiri, and Pearl 1991)). It remains to show that the reftimes satisfy the timepoint bounds. It has already been noted that the upper bound is satisfied since for each timepoint x, distance(refpoint, x)  $\leq$  distance(origin, x), because of the 0 edge from the refpoint to the origin.

Next we show that distance(refpoint, origin)  $\geq 0$ . Otherwise, suppose there is a shortest path from the refpoint to the origin that is negative. Let y be the first timepoint (after the refpoint) on the path. Then reftime(y) = preftime(y) since the shortest path from the refpoint is the direct edge. We also have distance(refpoint, y) + distance(y, origin) < 0. Thus,

$$preftime(y) = reftime(y)$$
  
= distance(refpoint, y)  
< -distance(y, origin)  
= lower-bound(y)

which contradicts the definition of preftime(y).

Thus, distance(refpoint, x) + distance(x, origin)  $\geq 0$ for each timepoint x. It follows that reftime(x)  $\geq$ lower-bound(x).

We see that the preftime is the distance along the direct edge to a timepoint x while the reftime is the shortest path distance to x. Thus, reftime $(x) \leq \text{preftime}(x)$ . The following result shows that among consistent schedules that satisfy this condition, the reftime schedule is the closest to the preftime.

**Theorem 3** The reftime schedule is maximal among schedules S such that S satisfies the temporal constraints and  $S(x) \leq preftime(x)$  for each timepoint x.

#### **Proof:**

Consider any such S and suppose  $S(x_1) > \text{reftime}(x_1)$ . Let  $x_2$  be the first timepoint (after the refpoint) on a shortest path from the refpoint to  $x_1$ . Then  $\text{reftime}(x_2) = \text{preftime}(x_2)$ .

By the temporal constraints, we have  $S(x_1) - S(x_2) \leq \text{distance}(x_2, x_1)$ . Thus,

$$S(x_2) \geq S(x_1) - \text{distance}(x_2, x_1)$$
  
> reftime(x\_1) - distance(x\_2, x\_1)  
= reftime(x\_2)

$$=$$
 preftime $(x_2)$ 

But this violates the condition of the theorem.  $\Box$ 

The following corollary shows the reftimes satisfy the most basic stability property.

**Corollary 3.1** If the preferred time schedule satisfies the temporal constraints, then the reference schedule is the preferred time schedule.

**Proof:** Immediate from the theorem after noting that no lower-bound corrections are needed for the preferred times if they satisfy the temporal constraints.  $\Box$ 

With this definition of the reference schedule, the reftimes decrease from the preferred times to satisfy the constraints unless they have to be increased to satisfy lower bound requirements. There is a symmetrical counterpart to the definition of reference times where the default is to rise instead of drop. For this alternative definition, preftimes are installed as lower bound constraints from the refpoint; for example, we would install a prefilme on x as a lower bound constraint  $x - refpoint \ge prefine(x)$  instead of x – refpoint  $\leq$  preftime(x). Similar results then hold, but with the roles of lower and upper bounds interchanged. However, the drop default seems preferable to the rise default since it tends to minimize makespan. One can construct examples where the drop variant perturbs the preferred times less than the rise variant and vice versa. The approach of (Morris et al. 2004) could minimize the sum of the perturbations, but at an added computational cost. The refpoint propagation method seems like a good compromise that combines efficiency and effectiveness.

In terms of complexity, the computation of the reftimes involves one extra Dijkstra propagation above the standard propagation in an STN. It should be noted that this has a somewhat different character than the lower/upper bound propagations. A tightening of the constraints may cause a lower bound to increase. This may cause the preftime to also increase, because of a lower bound correction. However, an increase in the preftime amounts to a *loosening* of the preftime constraint. The repropagation needed after a loosening typically affects more timepoints than after a tightening, so this is more costly in practice, though the theoretical complexity is the same.

Suppose E is the number of edges and N is the number of nodes in the STN. The discrete event simulation needed to detect the flaws is O(N). If we add the  $O(E + N \log N)$ cost associated with the additional Dijkstra propagation, this compares very favorably with the maximum flow method needed for flexible flaw detection, which is O(NE), or worse, for the common algorithms.

# **State-Derived Resources**

In the previous sections, we have described an approach that uses a temporally grounded reference schedule to detect a subset of the flaws in a flexible plan. Those flaws are resolved in the *flexible* plan by adding new *flexible* constraints. Then a new reference schedule is produced to satisfy the new constraints, and the process iterates.

Although developed in a context where the flaws are resource flaws, there is no obvious reason why the general approach could not also be applied to problems where the flaws arise from other sources, such as state requirements and effects. One advantage of applying this to state-based reasoning is that modal truth criterion issues are particularly simple for a temporally grounded schedule (Kambhampati and Nau 1994). Note that the search framework developed here makes no *commitment* to the ordering in the reference

Boolean Construct	Numerical Encoding				
Initial TRUE	1000				
Initial FALSE	0				
$TRUE \rightarrow FALSE$	Subtract 1000				
$FALSE \rightarrow TRUE$	Add 1000				
Start require TRUE	Subtract 1				
End require TRUE	Add 1				
Violation	< 0				

Table 1: Numerical encoding of Boolean state.

schedule; it merely uses it to detect and fix flaws in the associated flexible plan. Thus, it has the potential to sidestep the efficiency issue raised in (Kambhampati and Nau 1994, page 22), which relates to premature commitment.

As a first step towards applying this method to statebased reasoning, we have developed an encoding of state requirements and effects as resource transactions in a context of temporally extended activities and temporal constraints (Morris and Bresina 2008). We hope this will serve as a useful theoretical scaffolding for guiding the extension to state-based reasoning, as well as providing an initial vehicle for experiments. This is still a work in progress, but we discuss some of the implications here.

The encoding is initially applied to Boolean states where certain activities can reset the state from TRUE to FALSE or from FALSE to TRUE at their start or end timepoints and certain activities can require the TRUE state for their duration. A Boolean state is encoded using a consumable resource as follows. If the state is initially true, the initial capacity of the resource is a specific large number that we designate truevalue (currently 1000); otherwise the initial capacity is zero. An event that changes the state from TRUE to FALSE consumes truevalue amount of the resource, whereas an event that changes the state from FALSE to TRUE produces a like amount. An activity that requires the TRUE state consumes a unit amount of the resource at the beginning and returns it at the end. It is considered a violation if the available capacity of the resource drops below zero. The encoding is summarized in table 1.

In the TRUE state, this numerical encoding permits up to truevalue concurrent instances of activities that require the TRUE state, whereas in the FALSE state any occurrence of such an activity drops the numerical resource below 0, and thus produces a violation. Thus, truevalue should be set sufficiently large so that it can accommodate whatever amount of concurrency is needed for the application. There is no penalty for making it as large as one likes, subject to computer datatype limits. In table 1 and the remainder of the paper, we use 1000 for the truevalue amount.

As a first extension, the encoding can be generalized to enumerated state values provided each activity type determines a well-defined sequence of state transitions. For example, in planning for a Mars rover, a state variable for a robot arm might involve {UNSTOWING. UNSTOWED, STOWING, STOWED} as values and a STOW activity type might specify the sequence

UNSTOWED 
$$\rightarrow$$
 STOWING  $\rightarrow$  STOWED

where the transitions occur at the start and end timepoints, respectively, and UNSTOWED is regarded as a precondition.

For a model with these restrictions, the state requirements and effects can be translated into producer and consumer transactions on consumable resources. The translation uses a different numeric resource for each state value. For example, in the Mars rover case, there would be numeric resources for each of UNSTOWING. UNSTOWED, STOWING, STOWED, and



illustrates the effects of the STOW activity. Note that if the arm is already stowed, the value of the UNSTOWED resource will be 0, in which case this activity drops it to -1000, which is detected as a flaw.

A requirement only affects the resource for the particular state value that is required. For example a DEPLOY activity that requires the robot arm to be in the unstowed state would have the following resource transactions

DEPL	OY	
-1	+1	UNSTOWED

which subtract 1 at the start and add 1 at the end.

In earlier sections, we saw that the grounded resource solver algorithm (Figure 1) tries to resolve resource flaws by adding "culprit after savior" constraints. To illustrate the effect on state-derived resources, consider a flawed reference schedule with the following sequence of activities

STOW	DEF	PLOY	UNSTOW	
-1000	-1	+1	+1000	UNSTOWED

where the possible fixes are to constrain the -1000 culprit to follow the +1 savior, which makes DEPLOY come before the STOW, or the -1 culprit to follow the +1000 savior, which makes DEPLOY come after the UNSTOW. These alternatives are roughly analogous to promotion and "white knight" operations (Kambhampati and Nau 1994) in previous planning methods. Interestingly, demotion and causal linking are not needed in this restricted framework. Our past experience in planning for Mars rovers suggests that causal linking often produces undesirable premature commitments. For example, if two activities that require the same state have been linked to the same achiever, then a third activity that requires a different state cannot be inserted between them without backtracking.

We are exploring some further directions for extending the numeric resource encoding. One involves allowing state changes with unspecified prior states. A simple example of this is where the STOW activity is considered valid even if the arm is already STOWED. (In that case, it has no effect.) This can be handled by modifying the way in which the numeric resource level is updated so that it uses what might be called *saturated arithmetic* instead of ordinary arithmetic. Specifically, if a resource transaction (consumption or production) would otherwise cause the available resource level to fall outside the range [-999, 1000], then multiples of 1000 are added or subtracted to bring it back within that range. For example, two consecutive occurrences of the STOW activity would bring the level of the UNSTOWED resource to 0 instead of -1000. Saturation occurs naturally for some purely numerical applications, such as a FILL-UP operation for an automobile.

One implication of the use of saturated arithmetic is that addition is no longer commutative. For example, ((1000 - 1000) + 1000) = 1000 but ((1000 + 1000) - 1000) = 0. Because of this, Lemma 1 as stated no longer holds. For example, a reference schedule with the sequence

UNSTOW	STOW	DEPLOY		]	
+1000	-1000	-1	+1		UNSTOWED

has a flaw at the beginning of DEPLOY. One way of fixing this is to add a constraint that moves the -1000 culprit at STOW start beyond the +1 savior at DEPLOY end. The effect of this is to reorder the plan to be UNSTOW, DEPLOY, STOW. However, suppose an existing temporal constraint prevents this fix. Because of the non-commutative nature of saturated arithmetic, an alternative fix is to add a constraint that moves the -1000 culprit at STOW start so that it comes before the +1000 helper at UNSTOW end. Assuming STOW and UNSTOW are not allowed to overlap, this reorders the plan to be STOW, UNSTOW, DE-PLOY. If the arm was initially STOWED, then the STOW operation is now a no-op. If the arm was initially UN-STOWED, then the UNSTOW operation, which was previously a no-op, now becomes effective. This alternative fix method is roughly analogous to demotion (Kambhampati and Nau 1994) in conventional planning. Note that causal linking is still not needed.

For an example that does not involve ineffective operations, suppose the robot arm can be rotated to different positions to enable the use of different instruments, say a microscopic imager (MI), Mossbauer spectrometer (MB), or rock abrasion tool (RAT). Initially, it is in the RAT position. Consider the flawed plan

GO_MI		GO_MB	USI	E_MI	
	+1000	-1000	-1	+1	MI-AVAIL
-1000		-1000			RAT-AVAIL

where we have illustrated the effects on two of the resources. Since the prior state is unspecified for the GO\_MI and GO\_MB activity types, they decrement the resources for *all* the complementary states. Thus, each subtracts 1000 from RAT-AVAIL. Because of saturation, this does not constitute a flaw. However, the MI-AVAIL resource does have a flaw. The fixes for this are similar to the prior example, but the activities all have an effect before and after the fixes.

Although Lemma 1 as stated no longer holds with saturated arithmetic, we can prove a weaker result that involves moving a culprit either beyond a savior or before a prior helper. This change ripples through the other theoretical results. We get a modified algorithm that is still complete and correct but has additional fix options, and a flexible plan extraction method that requires preserving both producer  $\leq$  consumer and consumer  $\leq$  producer orderings from the reference schedule.

Note that consideration of saturated arithmetic is unnecessary for computing the resource flaws since the grounded reference schedule determines the prior states. The search and extraction methods are specified in terms of culprits, saviors, and helpers rather than arithmetic. We have used saturated arithmetic as a theoretical device to derive the modified search and extraction procedures and prove their soundness rather than incorporating it directly in the computational methods.

There is another way of approaching this issue that retains ordinary arithmetic. The idea is to enlarge the concept of a reference schedule to that of a reference *plan* that includes a reference grounding of the parameters as well as being temporally grounded.

We could regard the GO actions as having implicit *from* parameters. If we make these explicit, the plan becomes

GO\_MI\_FROM(RAT) GO\_MB\_FROM(MI) USE\_MI

where as before RAT is the initial position. Now the fully grounded activities specify well-defined state transitions. The flaw can be fixed by constraining the middle activity so that it comes first, but this requires simultaneously changing the *from* parameters of the first and second activities to arrive at the plan

GO\_MB\_FROM(RAT) GO\_MI\_FROM(MB) USE\_MI

With this interpretation, it is the modified parameters that have altered the resource computation to fix the flaw rather than a modified arithmetic. However, this seems to require a more complex analysis that tracks the resource effect of parameter changes. We have not pursued this approach.

# **Closing Remarks**

In the past, temporally flexible resource solvers have been handicapped by the complexity of performing inference within flexible time, not just in terms of performance, but also in terms of extensibility to more diverse resource types. Combining flexible solving with an evolving grounded schedule helps alleviate this by allowing resource flaws to be detected using a simple linear discrete event simulation. We have also introduced a new method for updating the grounded schedule that has a low computational complexity and satisfies a global stability criterion.

We have also discussed an encoding of state conditions and effects as resource transactions. An interesting aspect of active solving of state requirements using the resource encoding of (Morris and Bresina 2008) is that *causal links* (Weld 1994) seem not to be needed. This furthers the objective of least commitment planning, since causal links involve an arbitrary commitment to a particular achiever. The state encoding also motivated an extension of the resource model to consider saturated resources.

In the future, we would like to apply the grounded inference approach to active solving for more complex resource types like linear resources (Frank and Morris 2007), since flaws can be more easily detected in a grounded reference schedule. We would also like to develop a direct statebased solver that detects flaws based on a grounded reference schedule, but without the intervening resource encoding.

More generally, the idea of combining a least commitment search with a reference assignment to the uncommitted variables may be applicable to other types of planning and constraint satisfaction problems.

### References

Bresina, J. L., and Morris, P. H. 2007. Mixed-initiative planning in space mission operations. *AI Magazine* 28(2):75–88.

Cesta, A.; Oddi, A.; and Smith, S. F. 1998. Scheduling multi-capacitated resources under complex temporal constraints. In *Fourth Int. Conf. on Principles and Practices of Constraint Programming (CP98)*.

Cormen, T.; Leiserson, C.; and Rivest, R. 1990. *Introduction to Algorithms*. Cambridge, MA: MIT press.

Dechter, R.; Meiri, I.; and Pearl, J. 1991. Temporal constraint networks. *Artificial Intelligence* 49:61–95.

Frank, J., and Morris, P. 2007. Bounding the resource availability of activities with linear resource impact. In *International Conference on Automated Planning and Scheduling*.

Kambhampati, S., and Nau, D. S. 1994. On the nature and role of modal truth criteria in planning. *Artificial Intelligence* 82:129–155.

Laborie, P. 2003. Resource temporal networks: Definition and complexity. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence*.

Morris, P., and Bresina, J. 2008. Active and passive constraint enforcement for activity planning. In *International Symposium on Artificial Intelligence, Robotics and Automation in Space.* 

Morris, P.; Morris, R.; Khatib, L.; Ramakrishnan, S.; and Bachmann, A. 2004. Strategies for global optimization of temporal preferences. In *Tenth International Conference on Principles and Practices of Constraint Programming (CP-2004*, 408–422. Springer.

Muscettola, N. 2004. Incremental maximum flows for fast envelope computation. In *Proceedings of the 14th International Conference on Automated Planning and Scheduling*.

Weld, D. 1994. An introduction to least commitment planning. *AI Magazine* 15(4):27–61.

Yoon, S. W.; Fern, A.; and Givan, R. 2007. Ff-replan: A baseline for probabilistic planning. In *ICAPS*-07, 352–359.